Deepfake Image Detection

May 30, 2025

Task 2 - Deepfake Detection

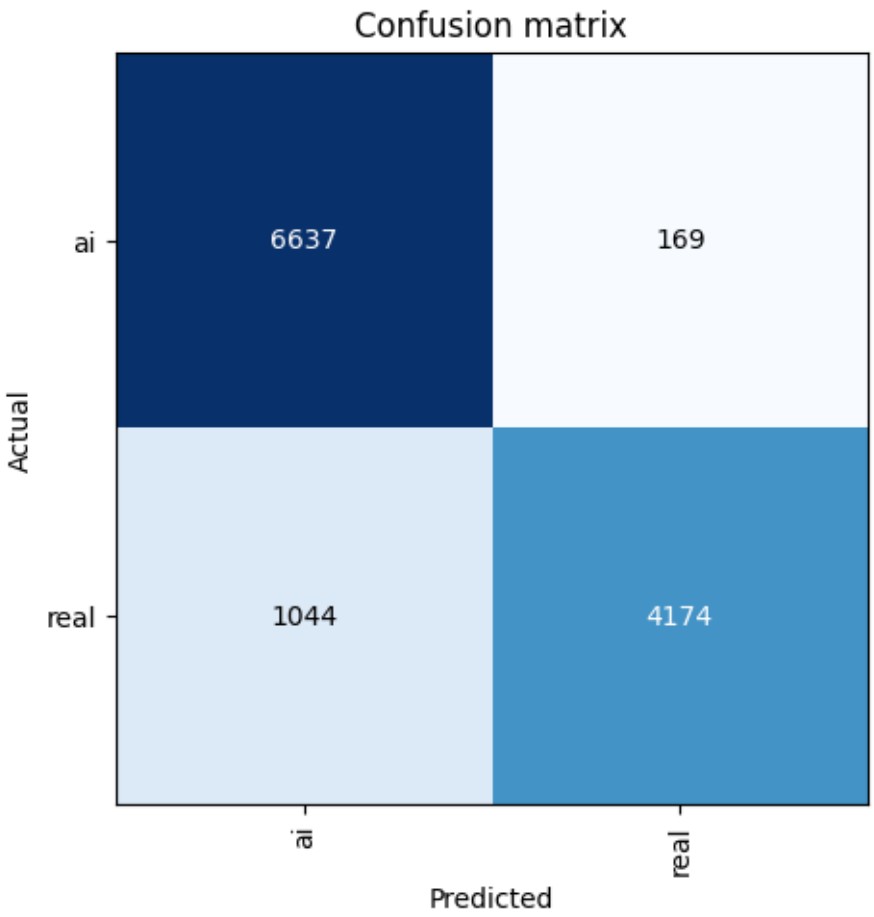
In this task, we will create a model to detect deepfake images using a dataset of real and fake images. Similar to the paddy disease detection task, we will leverage transfer learning and fine-tune a pre-trained model for this purpose.

Initial Experiments: ResNet50 Baseline

I began by analyzing the dataset and applied similar preprocessing techniques as used in the paddy project. For our baseline, I selected the ResNet50 architecture due to its proven performance and fast training capabilities, especially in early prototyping. By keeping the number of training epochs low, I could quickly iterate over various configurations of image size, augmentations, and learning rates.

To determine the optimal input resolution, I conducted experiments with image sizes of 512×512, 256×256, and 128×128. Surprisingly, I observed that smaller image sizes often yielded better performance, likely due to the model's improved ability to generalize. However, reducing the resolution too much caused the model to miss subtle artifacts and imperfections characteristic of deepfakes. Ultimately, the 256×256 resolution struck a balance between detail and generalization, producing the best results for ResNet50. This model achieved a Kaggle score of approximately 0.895, which served as a strong baseline.

epoch	train_loss	valid_loss	error_rate	time
0	0.193611	0.365707	0.112774	03:11
1	0.155723	0.326862	0.102295	03:11
2	0.125915	0.298038	0.095060	03:14
3	0.133721	0.297405	0.100882	03:19


Confusion matrix
ai
real
6637
169
1044
4174
Actual
ai
real
Predicted

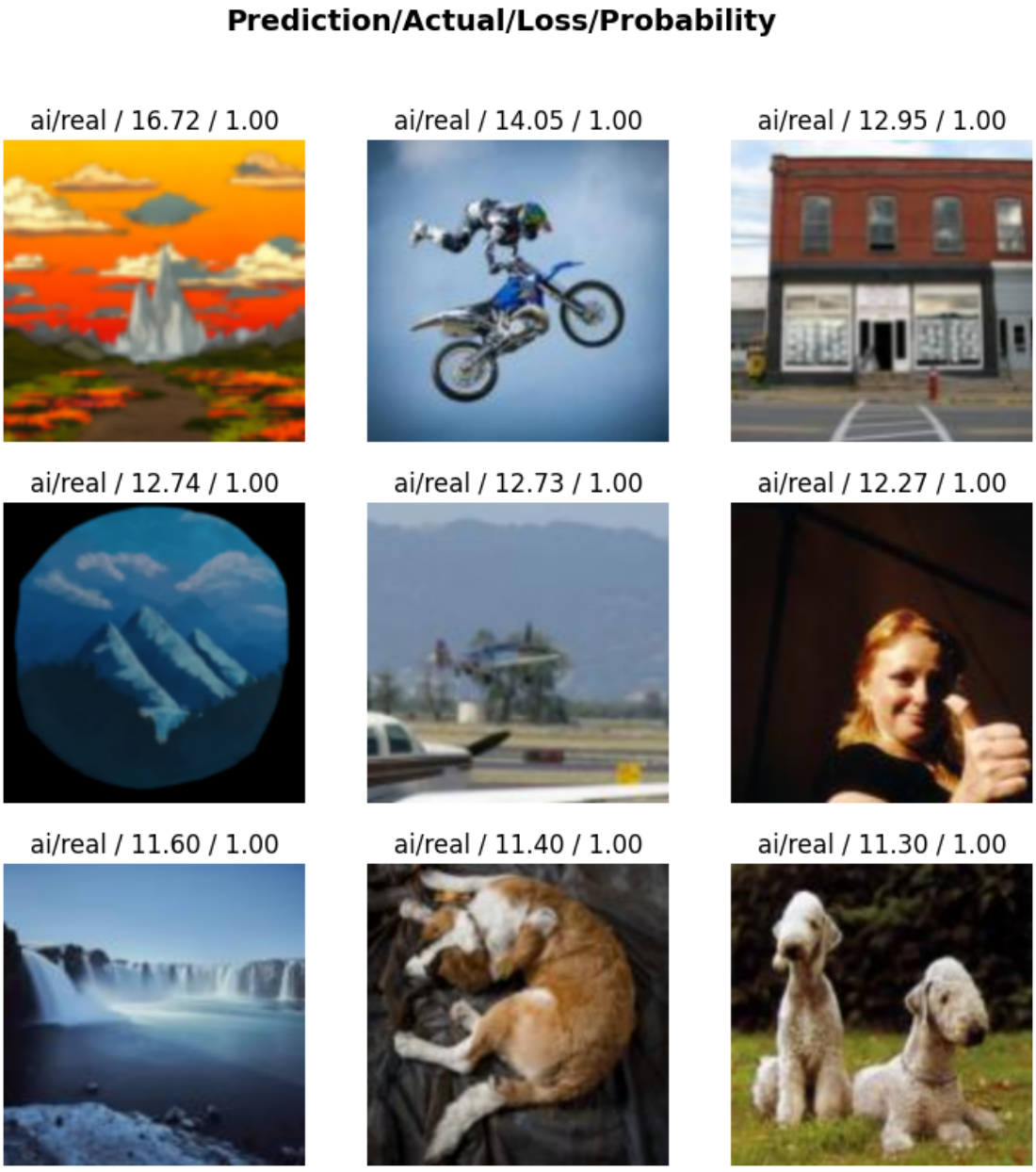


Figure: Images show confusion matrix and top losses for the best ResNet50 model

An interesting thing about the confusion matrix for the ResNet model is that it does not look very good. There is a lot of wrongly predicted images and especially when the image is real, but the model predict ai. Regardless of this not so good confusion matrix, it produces very good Kaggle scores, which tells me that the other models that I have tested likely have overfitted heavily on the dataset. The overfitting, however, seems also a bit strange as the dataset is quite large and I usually only run the models for a few epochs.

Testing Alternative Architectures

I then explored other architectures such as ConvNeXt Small and Vision Transformers (ViT), hypothesizing that their ability to model local textures could offer an advantage in detecting subtle deepfake imperfections. Multiple configurations were tested, including different patch sizes and training schedules.

Although the ConvNeXt Small model performed comparably well, reaching a score of 0.889, it required significantly more computational resources and longer training time without offering a meaningful improvement. Similarly, ViT models showed promise but consistently fell short of outperforming ResNet50. These results suggest that for this particular task, ResNet50 strikes a favorable tradeoff between accuracy, training speed, and generalization.

Ensemble Strategy

As in the paddy disease detection task, I attempted to ensemble multiple models with ResNet50, ConvNeXt Small, and ViT variants—to combine their strengths. Ensembling is generally expected to reduce variance and leverage complementary decision boundaries, leading to improved performance.

However, in this case, the ensemble approach did not lead to an improvement over the best single ResNet50 model. The averaged predictions showed minor fluctuations in accuracy and even underperformed slightly compared to the standalone ResNet50. Several factors may explain this; Firstly, rhe individual models may have learned similar decision boundaries, reducing the benefit of ensembling. Secondly, averaging softmax probabilities across models with different confidence levels might have introduced noise. And lastly, models like ViT and ConvNeXt, though powerful, might fail to pick up on the exact set of features critical for this task, weakening the ensemble's consensus.

[ ]:
```
!pip install fastkaggle fastai kagglehub
```

[ ]:
```
import timm

from fastkaggle import *

comp = 'hack-rush-deep-fake-detection'

path = setup_comp(comp, install='fastai "timm>=0.6.2.dev0"')
```

[ ]:
```
import kagglehub

train_path = kagglehub.dataset_download("shreyansjain04/
 ↪ai-vs-real-image-dataset")

test_path = kagglehub.dataset_download("shreyansjain04/
 ↪ai-vs-real-image-test-dataset")
```

[ ]:
```
from fastai.vision.all import *
```

```
trn_path = Path('mic')
tst_path = Path('mic-test')
```

[ ]:
```
resize_images(train_path, dest=trn_path, max_size=128, recurse=True,␣
↪max_workers=7)
```

[ ]:
```
resize_images(test_path, dest=tst_path, max_size=128, recurse=True,␣
↪max_workers=8)
```

Train function

[ ]:
```
def train(arch, size, item=Resize(480, method='squish'), accum=1,␣
↪finetune=True, epochs=12):
    dls = ImageDataLoaders.from_folder(trn_path, valid_pct=0.2, item_tfms=item,␣
↪batch_tfms=aug_transforms(size=size, min_scale=0.75), bs=64//accum)
    cbs = GradientAccumulation(64) if accum else []
    learn = vision_learner(dls, arch, metrics=error_rate, cbs=cbs).to_fp16()
    if finetune:
        learn.fine_tune(epochs, 0.01)
        learn.export(f"{arch}_{size}_e{epochs}")
        tst_files = get_image_files(tst_path)

        interp = ClassificationInterpretation.from_learner(learn)
        interp.plot_confusion_matrix()
        interp.plot_top_losses(9)

        test_dl = learn.dls.test_dl(tst_files)
        preds, _ = learn.tta(dl=test_dl)

        submission = pd.DataFrame({
            "filename": [f.name for f in test_dl.items],
            "class": preds.argmax(dim=1).numpy()
        })
        submission.to_csv("submission.csv", index=False)
        return learn
    else:
        learn.unfreeze()
        learn.fit_one_cycle(epochs, 0.01)
```

Ensemble functions

[ ]:
```
res = 128,128
models = {
    'convnext_small_in22k': {
        (Resize((128, 128)), 224),
        (Resize((256, 256)), 299),
        (Resize((128, 128)), 320),
```

```
    },
    'vit_small_patch16_224': {
         (Resize((128, 128)), 224),
         (Resize((256, 256)), 224),
    }
}
```

[ ]:
```
import gc
tta_res = np.load("tta_res.npy")

for arch,details in models.items():
    for item,size in details:
        print('---',arch)
        print(size)
        print(item.name)
        tta_res = np.append(tta_res, train(arch, size, item=item, accum=2,␣
 ↪epochs=10)) #, epochs=1))
        gc.collect()
        torch.cuda.empty_cache()
```

[ ]:
```
tta_prs = first(zip(*tta_res))
avg_pr = torch.stack(tta_prs).mean(0)
dls = ImageDataLoaders.from_folder(trn_path, valid_pct=0.2,␣
 ↪item_tfms=Resize(480, method='squish'),
    batch_tfms=aug_transforms(size=224, min_scale=0.75))

tst_files = get_image_files(tst_path)

idxs = avg_pr.argmax(dim=1)
vocab = np.array(dls.vocab)

submission = pd.DataFrame({
    "image_id": [f.name for f in tst_files.items],
    "label": vocab[idxs]
})
submission.to_csv("deepfake-submission.csv", index=False)
```

ResNet50 Model

[ ]:
```
import gc
gc.collect()
torch.cuda.empty_cache()
learn = train('resnet50', 256, item=Resize((128, 128)), accum=1, epochs=6)
```

[ ]:
```
if not iskaggle:
    from kaggle import api
    api.competition_submit_cli('submission.csv', 'resnet50 128 6e', comp)
```

Other methods from the literature

After researching the topic, I found a paper called "Fighting deepfake by exposing the convolutional traces on images" (Guarnera et al., 2020) that suggested using a technique were you calculate convolution traces of the images, by using a algorithm called Expectation Maximization (EM). The method works by calculating the convolution traces of the images, which are the tiny patterns, imperfections and artifacts left behind by the GANs (Generative Adversarial Networks) that create deepfakes. These traces are often too subtle for the human eye to detect, but they can be captured by the EM algorithm. The EM algorithm iteratively refines the estimates of the convolution traces until they converge to a stable solution. Once the traces are calculated, they can be used as features for a classifier, such as Random Forest, to distinguish between real and fake images. A simple implementation of this algorithm was tested, and by using the Random Forest classifier, as recommended in the paper, I managed to get a accuracy of around 0.6, which is not particularly good. I therefore tried to use Support Vector Classification (SVC) as an alternative to Random Forest classifier. This increased the accuracy to 0.7, and shows that the method works, as the computed convolution traces have some signal, but they are noisy and hard to classify. In the paper, they get an even higher accuracy at around 0.9 and above, which tells me that my implementation of the paper might not be as good as theirs, and without GPU acceleration of the convolution trace computation it takes many hours to finish the dataset.

Looking at this solution it is quite nice to see that the task is solvable without needing a black box, which these CNNs and RNNs are, and it is possible to explain how the model made its prediction on a understandable level. Making an understandable model will be important to be able to gain trust in the model and will supersede any deep learning model with equal accuracy.

Another paper titled "Deepfake Detection and Classification of Images from Video: A Review of Features, Techniques, and Challenges" (Bale, et al. 2024), gives a structured review of how deepfake images, especially those extracted from video, are detected and classified and it outlines three main approaches.

Feature-based methods rely on spotting visual inconsistencies like unnatural lighting, irregular eye reflections, or distorted facial expressions. Traditional machine learning models use predefined features to train classifiers, but they often struggle to adapt to new types of deepfakes. Deep learning techniques, and especially convolutional neural networks (CNNs) are more robust, as they can learn complex features and adapt to small manipulations.

The paper points out several ongoing challenges in the field, such as the rapid evolution of deepfake creation tools, the limited availability of diverse and representative datasets, and the difficulty of building models that perform well across different scenarios. To address these, the authors propose a framework for comparing detection techniques and stress the importance of real-world applicability, model robustness, and future research aimed at staying ahead of increasingly realistic forgeries.

Guarnera, L., Giudice, O., & Battiato, S. (2020). Fighting deepfake by exposing the convolutional traces on images. IEEE access, 8, 165085-165098.

Bale, D. L. T., Ochei, L. C., & Ugwu, C. (2024). Deepfake Detection and Classification of Images from Video: A Review of Features, Techniques, and Challenges. International Journal of Intelligent Information Systems, 13(2), 17–27. https://doi.org/10.11648/j.ijiis.20241302.11

Convolutional traces

```
[ ]: import cv2
import numpy as np

def extract_ct(image, kernel_size=3, max_iter=10):
    def em_channel(channel):
        alpha = kernel_size // 2
        padded = cv2.copyMakeBorder(channel, alpha, alpha, alpha, alpha, cv2.
 ↪BORDER_REFLECT)
        h, w = channel.shape
        N = h * w
        d = kernel_size**2 - 1

        patch_offsets = []
        center = kernel_size // 2
        for i in range(kernel_size):
            for j in range(kernel_size):
                if i == center and j == center:
                    continue
                patch_offsets.append((i - center, j - center))

        A = np.zeros((N, d), dtype=np.float32)
        b = np.zeros(N, dtype=np.float32)

        idx = 0
        for y in range(alpha, h + alpha):
            for x in range(alpha, w + alpha):
                A[idx] = [padded[y + dy, x + dx] for dy, dx in patch_offsets]
                b[idx] = padded[y, x]
                idx += 1

        k = np.zeros(d, dtype=np.float32)
        for _ in range(max_iter):
            pred = A @ k
            residuals = b - pred
            sigma2 = np.mean(residuals**2)
            weights = np.exp(-residuals**2 / (2 * sigma2))

            Aw = A * weights[:, np.newaxis]
            bw = b * weights
            k = np.linalg.pinv(A.T @ Aw) @ (A.T @ bw)

        return k

    if image.shape[2] != 3:
        raise ValueError("Image must be RGB")
```

```
    image = image.astype(np.float32) / 255.0
    return np.concatenate([em_channel(image[..., c]) for c in range(3)])
```

[ ]:
```
import numpy as np

from fastai.vision.all import *
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from pathlib import Path

def get_balanced_subset(path, max_per_class=500):
    files = get_image_files(path)
    grouped = {}
    for f in files:
        lbl = parent_label(f).lower()
        grouped.setdefault(lbl, []).append(f)

    selected = []
    for lbl, f_list in grouped.items():
        selected.extend(f_list[:max_per_class])

    return selected

path = Path("sml")

dblock = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    get_y=parent_label,
    splitter=RandomSplitter(seed=42)
)

dls = dblock.dataloaders(path, bs=16)

from tqdm import tqdm

i = 0
X, y = [], []
for img, label in tqdm(dls.train_ds):
    try:
        ct_vec = extract_ct(np.array(img), kernel_size=3)
        X.append(ct_vec)
        y.append(int(label))
    except ValueError:
        print(img)
        print("Image not RGB")
```

```
    i += 1
    if i % 10000 == 0:
        np.save(f"ct_vectors_{i}.npy", X)

X_valid, y_valid = [], []
for img, label in tqdm(dls.valid_ds):
    try:
        ct_vec = extract_ct(np.array(img), kernel_size=3)
        X_valid.append(ct_vec)
        y_valid.append(int(label))
    except ValueError:
        print("Image not RGB")


rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

y_pred = rf.predict(X_valid)
print(classification_report(y_valid, y_pred, target_names=["ai", "real"]))
```